Class Design

Grupo de autoestudio OCP - GuateJUG

Entendiendo herencia (inheritance)

- **Java solo soporta herencia simple**
- **Existe una jerarquía de clases en Java, en la que todas las clases heredan de** `java.lang.Object`
- **Métodos como** `toString()` **o** `equals()`**, entre otros, están disponibles desde** `java.lang.Object`
- **Uso de la palabra reservada** `extends` **para indicar que una clase** `B` **hereda de una clase** `A`

```
public class B extends A { }
```

Creando clases

- En cada archivo .java puede haber como máximo una clase de nivel superior (top-level class)
- Al declarar un tipo de nivel superior (top-level type) solo se puede usar el modificador package
- En un archivo .java, solo la clase de nivel superior puede hacer uso del modificador `public`, en cuyo caso debe ser nombrado como el archivo

Declarando constructores

- **El nombre del constructor coincide con el nombre de la clase**
- **En un constructor no se define tipo de retorno**
- **Si no se define un constructor de manera explícita, Java definirá implícitamente uno por defecto, de acceso `public`, sin parámetros**

```
public Foo() { } // para la clase Foo
```

- **Un constructor puede llamar a otro de la misma clase por medio de `this( ... )`**

Declarando constructores

- **Un constructor puede llamar a otro de la clase padre por medio de** `super( ... )`
- **El llamado a** `this( ... )` **o a** `super( ... )` **debe ser la primera sentencia de un constructor**
- **Si se define un constructor con parámetros de manera explícita, Java no definirá el constructor sin parámetros**

Inicializando objetos

- Los miembros estáticos en la jerarquía de clases se inicializan primero, siguiendo el orden Superclase > Subclase
- Luego las declaraciones de variables estáticas
- Luego los inicializadores estáticos
- Campos `final` de instancia (deben ser asignados una vez en el flujo del programa, ya sea cuando son declarados, en un inicializador o en un constructor)

Inicializando objetos

- Luego se inicializan las instancias:
  - Superclase > Subclase
  - Variables de instancia
  - Inicializadores de instancia
  - Constructor, incluyendo aquellos declarados mediante `this()`

Heredando miembros

- En la sobrescritura de métodos, un método en la clase hija:
  - Debe tener la misma firma que el método de la clase padre
  - Debe ser igual o más accesible que el método de la clase padre
  - No debe declarar una nueva excepción fuera del control del programa (checked exception) o una excepción más amplia que la definida en la clase padre
  - Si el método retorna un valor, debe ser del mismo tipo o subtipo de retorno del método de la clase padre (tipos covariantes)

Heredando miembros

- En los métodos estáticos no aplica el concepto de sobrescritura, sino el de ocultamiento (hiding)
  - En una clase hija se define un método con el mismo nombre y la misma firma que en la clase padre
  - Debe ser estático si el mismo método en la clase padre lo es
  - Aplican las mismas reglas de acceso y de tipos vistas con anterioridad

Heredando miembros

- **Otras consideraciones:**
  - **Los métodos de una clase padre declarados con el modificador `final` no pueden ser sobrescritos por las clases hijas**
  - **En las variables, al igual que en los métodos estáticos, no aplica el concepto de sobrescritura, sino el de ocultamiento (hiding)**

Creando clases abstractas

- **Una clase abstracta no puede ser instanciada directamente**
- **Una clase abstracta puede tener métodos abstractos y no abstractos**
- **Métodos abstractos se declaran sin cuerpo**
- **Métodos abstractos solo pueden ser declarados en clases abstractas**
- **Una clase hija no abstracta debe implementar los métodos abstractos que herede de la clase abstracta padre**
- **El modificador** `abstract` **es incompatible con** `final`, `private` **y** `static`

Creando objetos inmutables

- **Un objeto inmutable no cambia su estado después de su creación**
  - **Clase** `final` **o constructores** `private`
  - **Variables de instancia** `private` **y** `final`
  - **Sin *setters***
  - **Constructores con parámetros para establecer todas las propiedades del objeto**
  - **El objeto retornado al llamar un método es una copia defensiva, diferente al objeto establecido como argumento del método**

Repaso de preguntas

¡Hasta la próxima!